#include<iostream> → preprocessor directive. includes a file that our program needs later on.

using namespace std; → standard library. includes a library (a bunch of C++ stuff or basic words) that our program use

Every program starts with a function main.

Every statement ends with a semicolon inside function.

{ cout → output stream object.
<< → stream insertion operator.

return 0; → return statement.

main function must always have a return 0
It means the program terminated successfully.

<< endl; → new line
or inside the quotations "\n"

Naming a variable →

type name = value
eg int x = 6

(a) int a; → declaring
a=6 → assignment.

cin → input stream object
>> → stream extraction operator

```
# include <iostream>
using namespace std;
int main()
{
    int a, b, sum;
    cout << "Enter the two numbers";
    cin >> a >> b;
    sum = a + b;
    cout << "The sum is " << sum;
    return 0;
}
```


Precedence:

(i) Parenthesis
(ii) multiplication and division
(iii) addition and substraction.

If statement

if dies not satisfy no code.

syntax:

```
if (test condition)
{ code to run if test is true }
```

Single equal sign → assignment
double equal sign → for equality
eg in writing

!= → not equal to.

Relational operators > < <= >=

a>b a<b .
53>76
53>a etc

void → do not return anything.

If using another function outside the main function, first write the return type while creating a function tell the compiler first that you have created

eg int main()
{ print ( );
return 0; }
void print ( ).
{ cout << "Hi";
}.

X

It should be as,

```
void print();
{ cout << "Hi"; }

int main()
{ print();
  return 0; }
```

If you want the main at the type → use function prototype

```
void print();   → This prototype means that there is a function named print coming ahead so the compiler finds it itself and does not return an error.
int main()
{ print();
  return 0; }
void print()
{ cout << "Hi"; }
```

Progm

```
void print(int n)
{ cout << "My function no is " << n << endl; }

int main()
{ print(20);
  return 0; }
```

Function that requires two parameters.

```
#include <iostream>
Using namespace std;
int addnumbers (int x, int y)
{ int answer = x+y;
return answer; }

int main ( )
{ cout << addnumbers (2,4);
return 0;
}
```

OR if taking from user

```
int main ( )
{ int x, y;
cout << "Enter the numbers";
cin >> x >> y;
cout << "The sum is" << addnumbers(x,y);
return 0;
}
```

Classes and objects.

A group of similar functions → class.

For creating a class you have to se outside the main function

When public class → function can se used outside the class whereas if private it cannot se used outside the class eg by the main function etc.

object is how you access your stuff inside the class.

Program #include iostream using namespace std;


```
Class Salaar{
    public:
    void functionname ( )
    { cout<<"Rowan Regis"; }
};
int main ( )
{ Salaar sclass object;
  Salaar object . function name ( );
  return 0;
}
```

Using variables in classes

```
#include <iostream>  #include <string>
        using namespace std;
Class Backysclass
{       public:
  void setname (string x)
        { name = x; }
  string getname ()
        { return name; }
  private:
    string name;
        };
int main ()
{   Backysclass bo;
bo.setname ("Sabir Khan");
    cout << bo.getname ();
        return 0;
        }
```

while loop.

Syntax
while (test)
{ statement }.

eg Infinite loop with while

```
#include using namespace std;
{ int x=0;
while (x<=5)
{ cout<<"Hello world";}
return 0; }
```

To stop at a particular point.

```
int main()
{ int x=0;
while (x<=5)
{ cout<<"Hello world"<<endl;
x=x+1;}
return 0; }
```

Enter five different numbers, sum them and print them.

Program

```
#include <iostream>  Using namespace std;
int main()
{   int x=1;   → for the loop to run five times
    int number;  int total=0;
while (x<=5)
  { cin >> number;
    total = total + number;
    x++;
  }
cout << "Your total is " << total << endl;
    return 0; }
```

Sentinal Controlled Program

When we dont know how many times the loop has to run.

eg Program

```
#include <iostream>  Using namespace std;
int main()
{   int age;
    int agetotal;
    int numberofPeopleEntered;
```

```
cout<< "Enter age of first person or -1 to quit"<<endl;
cin >> age;
while (age != -1)
{ agetotal = agetotal + age;
  numberofpeopleentered ++;
cout<< "Enter the next age or -1 ";
cin >> age;
}
cout<< "Number of people entered" << numberofpeopleentered<<endl;
cout<< "The average age is" << agetotal / Number of people entered
                                                          <<endl;
return 0;
}
```


Assignment And Increment Operators.

$x = x + 10 \iff x += 10$

return 0 → not necessary in the new versions of compilers → it assumes returned 0.

$x -= 5 \iff x = x - 5$

$x = x \% 3 \iff x \%= 3$

$x = x / 3 \iff x /= 3$

Mt x=20

cout << x++ << endl;
cout << x << endl;

output

20
21

whereas

cout << ++x << endl;
cout << x << endl;

output

21
22

{

For loop

Syntax

for (initialization; ending value; increment)
↓
loop continuation condition.

Eg

```
int main()
{
for (int x=1; x<10; x++)
    cout << x << endl;
}
```

output

1
2
3
4
5
6
7
8
9.

Program

```
#include <iostream>   #include <cmath>
using namespace std;
int main()
{
    float a; float p=1000; float r; 0.01
for (int day=1 ; day<=20 ; day++)
{   a= p + pow(1+r, day);
    cout << day << "-----" << a << endl;
}
}
```


a → final amt ~~principal~~ amt mesta

r → profit rate each day

formula for final amt $a = p * (1+r)^{day}$

Do while loop.

```
int main()
{   int x = 1;
    do { cout << x << endl;
         x++; }
    while (x < 10);
}
```

In this loop it will print a set of code and then check the condition which means at least once the code is always printed in this loop

Switch Statement:

```
Switch (variable to be tested)
    { body }.
```

Example let variable is age.

```
int main ()
{       int age = 21;
    switch (age)
{   case 16:
      cout << " you can drive now " << endl;
        break;
    case 18:
      cout << " Id card " << endl;
        break;
    case 21:
      cout << " Hi how are you " << endl;
    defa  break;
        default:
      cout << " Sorry " << endl; }
```

break → to go out of a loop.

Switch takes a variable and tests it against its cases.

if its neither of the cases it goes to the default.

Logical operators.

```
int main()
{
    int age=23;
    int money=650;
    if (age>21)
    { if (money>500)
        { cout<<"you are a model";}
    }
}
```


Using a lot of if statements can be confusing.

```
{ int age=23;
  int money=650;
if (age>21 && money>500)
    { cout<<"you are a model";}
}
```

and && or || not !

Default argument Parameters.

```
#include <iostream>  Using namespace std;
int volume (int l, int w, int h);
int main ()
{   cout << volume (l, w, h) ; }
int volume (int l, int w, int h)
   { return l * w * h; }
```

(l, 5)

→ put the default value in the function prototype.


```
int x = 69;
int main ( )
{ int x = 20;
   cout << x << endl; }.
```

output 20 → local variable
if cout << ::x → output 69
→ global variable

Use a variable

when create a variable inside a function you can use it only in that function. when created outside a function. it can be used anywhere in a program.

If we have two variable with the same name (local and global) use :: unary scope operator to use the global variable.

Function Overloading

Two functions with the same name but for different datatypes.

eg.

```
void printNumber(int x)
{ cout<<"I am printing an integer "<<x<<endl; }

void printNumber(float x)
{ cout<<"Printing a float "<<x<<endl; }

int main()
{ int a=54;
  int b=32.489;
  printNumber(a);
  printNumber(b);
  return 0; }
```

Recursion → function can call itself.

base case → like end of function.

Array An array is basically a variable that can store multiple values

Syntax

Datatype name [size] = {, , , , , , }

values → array elements

10	20	30	2	4	68	→ elements
0	1	2	3	4	5	→ indexes

eg

```
int main ()
{ int Salary [5] = {1, 2, 3, 4, 5};
  cout << Salary [2]; }
```

output is 3

Creating Arrays Using loops:

```
Mt main ()
{ Mt Salary [9]
  cout << " Element == Value ";
```

```
for (int x=0; x<=8; x++)
    cout << x << endl;
}
```

output

0 1 2 3 4 5 6 7 8

(without curly braces)

```
for (int x=0; x<=8; x++)
    Salary[x]=99;
cout << x << ":" << Salary[x] << endl;
}
```

Using arrays in calculations.

```
#include <iostream> Using namespace std;
int main()
{ int Salary[5]={20,54,76,80,50};
  int sum=0;
  for (int x=0; x<5; x++)
  { sum = sum + Salary[x];
    cout << sum << endl;
  }
}
```

This is a program for calculating sum of elements

of the array where x stars the index.

Passing arrays to Functions.

```
#include <iostream>   #include <conio.h>
using namespace std;
void PrintArray( int thearray [], int SizeofArray );   → prototype
int main ()
{  int Scalar [3]= {20, 54, 675};
   int Amna [6] = {54, 24, 7, 8, 9, 99};
   printArray (Amna, 6) ;   (Scalar, 3)
}
void printArray (int theArray [], int SizeofArray)
{ for (int x=0; x< Sizeof array; x++)
   cout<< theArray[x] <<endl;
}
}
```

output

54, 24, 7, 8, 9, 99.

30, 54, 675.

Multidimensional Array

→ array inside another array
→ rows and columns

Syntax

Datatype Name [No. of rows][No. of cols] =
{ {1st row}, {2nd row}, {3rd row} };

eg: int main()
{ int Roman[2][3] = { {2,3,4}, {8,9,10} };
cout << Roman[1][1];
}

output is 9.

If cout << [0][0] → output is 2
If cout << [1][2] → output is 10 etc

How to print multidimensional array?

```
#include <iostream> Using namespace std;
int main()
{ int Maleeha[2][3] = {{1,2,3},{7,8,9}};
  for (int row=0; row<2; row++)
  { for (int column=0; column<3; column++)
    {cout << Maleeha[Row][Colum] << "--";
    }
    cout << endl;
  }
return 0;
}
```

Introduction of to Pointers

Pointer is a variable which stores the memory address as its value.

And sign & → address operator.

If you want to print the memory address of a variable let say x, write the "and" sign and then the variable as;

```
int x=5;
cout << &x;
```

```
eg  int main()
{   int fish = 5;
    cout << & fish << endl;

    int * fish pointer;
    fish pointer = & fish;
    cout << fish pointer << endl;

        return 0; }
```

Passing By Reference

passing by value → making each copy of variable
passing the copy of that variable

passing by reference → passing the memory address of the variable

eg program

```
#include <iostream> using namespace std;

void passByValue (int x);
void passByReference (int &x);

int main()
{   int betty = 13;
    int sandy = 13;
```

Pass by Value (Betty); pass by Reference (&Sady);
cout << "Betty is now " << Betty << endl;
cout << "Sady is now " << Sady << endl;

void PassByValue (int x)
{ x = 99; }

void passbyReference (int &x)
{ &x = 66; }

output
Betty is now 13.
Sady is now 66.

"Size of" Function

integer occupies 4 bytes of space.
character " 1 byte.
double " 8 bytes
↳ more precise.
The more precise a value is, the more memory it takes up.

eg let us any named Bucky.

int main ()
{ double Bucky[10];
cout << sizeof (Bucky); }

output = 80
B/c 1 double = 8
So 10 dbs = 80 bytes.

If cout << sizeof(bucky)/sizeof(bucky[0]);
↳ will give ~~size~~ number of elements in array.
An array has elements of the same datatype.
→ indication to the datatype

let

```
int main()
{ double c;
  cout << sizeof(c) << endl; }
```

o/p → 8.

Pointers And Math

```
int main()
{ int bucky[5];
  int *bp0 = &bucky[0];
  int *bp1 = &bucky[1];
  int *bp2 = &bucky[2];

cout << "bp0 is at " << bp0 << endl;
cout << "bp1 is at " << bp1 << endl;
cout << "bp2 is at " << bp2 << endl;
}
```

o/p

bp0 is at ...0
bp1 is at ...4
bp2 is at ...8

with pointers when added, it will not change the memory address it changes the value (pointed) it is pointing

e.g.

spo += 2; 0+2 → 2nd index

cout << "spo is now at " << spo << endl; → not like adding maths addition. It will not be as 1/p0 is not... 2.

output

spo is at 0... 0
sp1 is at --- 4.
sp2 is at 8-
spo is now at 8

Arrow Member Selection Operator:

.h → prototypes .cpp → coding

Deconstructors is a ~~thing~~ code that runs up itself upon the destruction of object.

Constant

datatype → keyword → variable is constant and it cannot be modified.

Function template

↳ a general datatype that can store multiple datatypes.

eg ~~int main()~~

```
template <class Bucky>
Bucky addcap (Bucky a, Bucky b)
  { return a+b };
int main()
{ int x=7, y=43 z;
  z = addcap(x,y);
  cout << z << endl;
}
```

Function Templates with multiple Parts

↳ different datatype

eg int & float

eg

```
template <class FIRST, class SECOND>
FIRST smaller (FIRST a, SECOND b)
{ return (a<b ? a : b); }

int main()
{
  int x=84;
  double y=56.78
```

cout << smaller (x, y) << endl;
}

output 56-

cout << smaller (y, x) << endl;

output 56-89

Traversing Sorting Searching Merging

Linear Search (Sequential Search)

```
#include <iostream>  #include <conio.h>  using namespace std;
int linearSearch (int array[], int size, int searchValue)
{ for (int i=0; i< size; i++)
{ If (searchValue == array [i])
{ return i;}
}
return -1; }
int main ( )
{ int a[ ] = {15, 23, 7, 45, 87, 16};
int userValue;
cout << "Enter an integer" << endl;
cin >> userValue;
```

```
int result = linearSearch (0, 6, userValue);
if (result >= 0)
{ cout << "The number " << a[result] << " was
    found at the element with index " << result << endl;
else
{ cout << "The number " << userValue << " was not
    found." << endl;
getch ();
}
```

Tracing

```
#include <iostream>  using namespace std;
double average (int array[], int size)
{ double sum = 0.0;
for (int i=0; i<size; i++)
  { sum = sum + array[i]; }
return sum/size; }
int main ()
{ int points[] = {18, 20, 5, 7, 22};
cout << average (points, 5) << endl;
}
```

Binary Search

→ we have to have a sorted array.
→ Eliminate half of the searchSpace.
→ comparison with midpoint value.
→ No. of comparisons = $\log_2$ (no. of elements)

Program

```
#include <iostream>. using namespace std;
int binarySearch (int array[], int size, int searchValue)
{ int low=0;
  int high = size-1;
  int mid;
  while (low<=high)
{ mid= (low+high)/2;
  if (searchValue == array[mid])
    { return mid; }
  else if (searchvalue > array[mid])
    { low = mid+1; }
  else { high= mid-1; } }
int main()
```

```
{ int a[ ] = {12, 22, 34, 47, 55, 67, 82, 98 2.
    int userValue;
cout << "Enter an integer" ;<<endl;
cin >> uservalue;

int result= binarySearch (a, 8, uservalue);
   if (result >=0)
{ cout<<"The number" << a[result] << "was
found at the index" << result << endl;
   }
   else
{ cout << "The user value << "was not found" <<endl; ;}
}.
```

Strings:

String constant. " " eg "I love c++"

Character constant. ' ' eg 'c'.

```
#include <cstring>
#include <iostream>
using namespace std;

int main()
{
    char x[] = "Happy Birthday";
    char y[25];
    char z[15];
    strcpy(y, x);

    cout << "The string in array x is " << x
         << "\n The string in array y is " << y << endl;

    strncpy(z, x, 14);

    cout << "The string in Array z is " << z << endl;

    return 0;
}
```

+ 10 (one space + terminating character)

copy

length → first 14 characters

concatinate.

```
int main()
{ char S1[20] = "Happy";
  char S2[] = "New year";
  char S3[] = "  ";
cout << "S1 = " << S1 << "\nS2 = " << S2;
strcat(S1, S2);
cout << "\n S1 = " << S1 << "\n S2 = "
     << S2;
strn cat(S2, S1, 6);
cout << " S1 = " << S1 << "\n" << "S3 = "
     << S3;
strcat(S3, S1);
cout << "S1 = " << S1 << endl;
cout << "S3 = " << S3 << endl;
return 0;
}
```

Strings compare.